Object Oriented Programming with Java 8
JDBC

Akshita Chanchlani

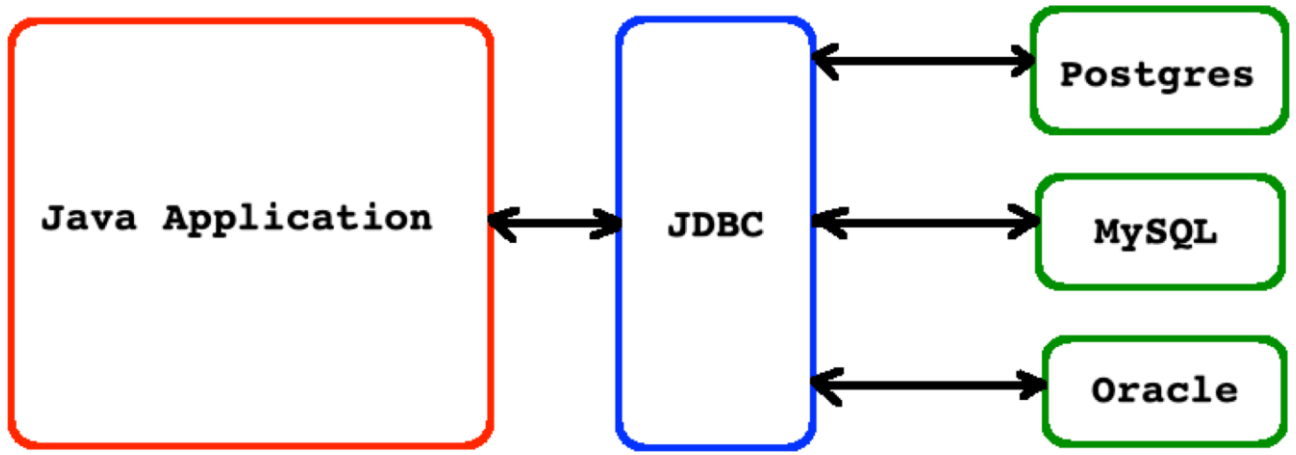
Java Database Connectivity

- Specification = { Abstract Class(es) + Interface(s) }
- JDBC is a specification:
  - Vendor : **SUN/oRACLE (java.sql package)**
  - Implementation : **Database vendors (Database Connector)**
  - Client : **Java Application Developers (Include connector in build path)**
- JDBC API version:

1	JDK 1.1	No Tag
2	J2SE 1.2	JDBC 2.0 API
3	J2SE 1.4	JDBC 3.0 API
4	Java SE 6	JDBC 4.0 API
5	Java SE 7	JDBC 4.1 API
6	**Java SE 8**	**JDBC 4.2 API**

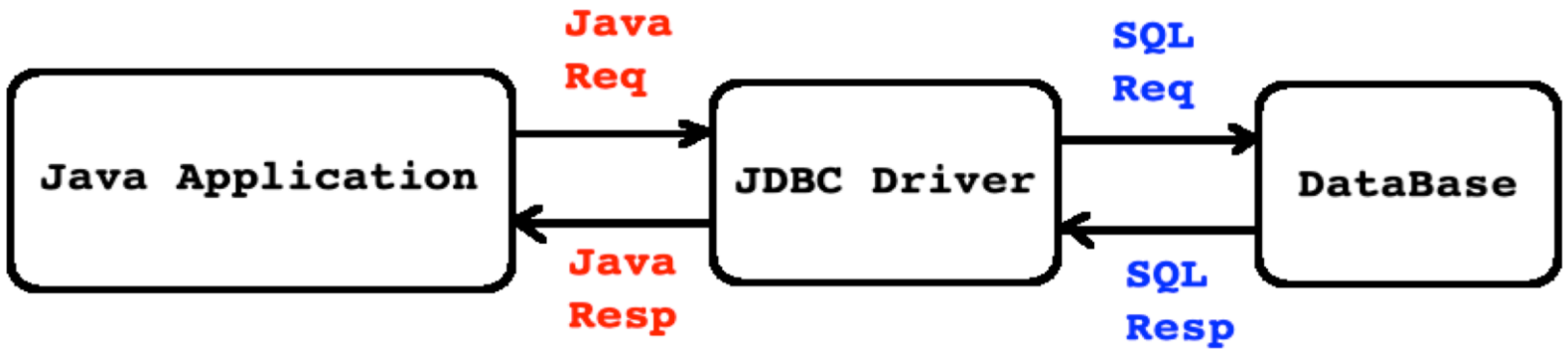
Why JDBC?

- If we want to access and process data stored in relational database management system using Java programming language then we should use JDBC.
- It minimizes database vendor dependency in the Java application.



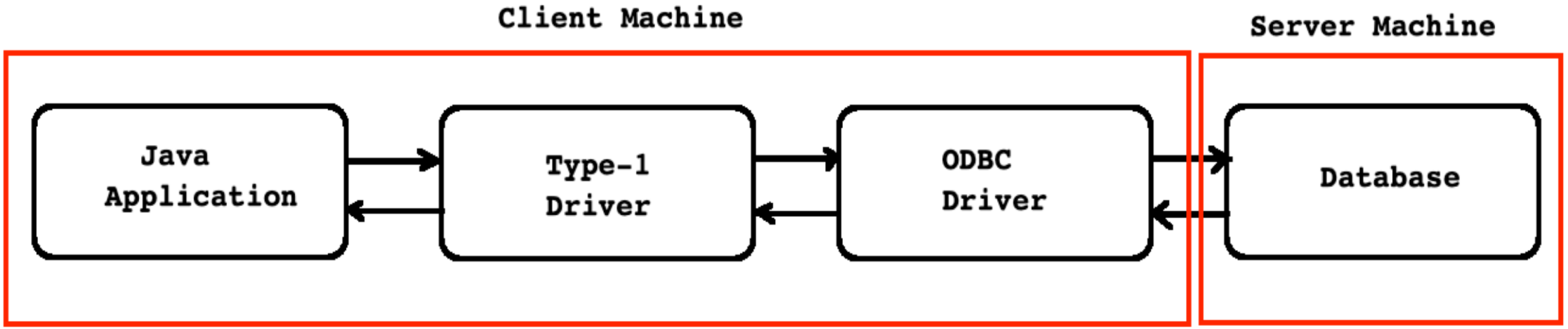
JDBC Driver

- Driver is a program, which converts JDBC API Calls(Java Request) into database specific calls(SQL Request) and vice versa.



- **on the basis of functionality and architecture there are 4 types of driver available:**
  1. Type – 1 Driver (JDBC-oDBC Bridge Driver)
  2. Type – 2 Driver (Native API Driver)
  3. Type – 3 Driver (Network Protocol / Middleware Driver)
  4. Type – 4 Driver (Database Protocol / Thin Driver)
- **Progress Data direct company provides Non standard Type – 5 Driver.**

JDBC Type 1 Driver Architecture



Type-1 Driver : To convert JDBC call into ODBC call or vice versa.

ODBC Driver : To convert ODBC call into database specific call or vice versa.

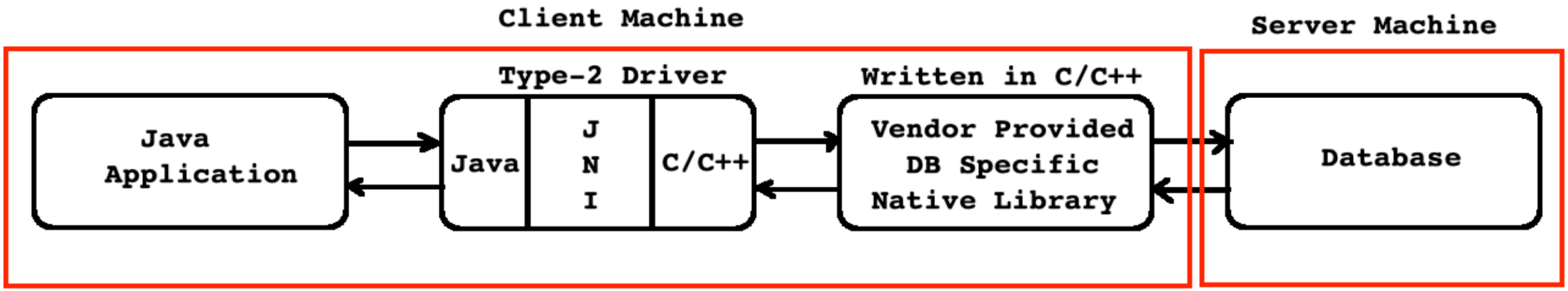
- Type – 1 driver is also called as JDBC-ODBC bridge driver.
- Example : sun.jdbc.odbc.JdbcOdbcDriver

JDBC Type-1 Driver

- **Advantages:**
  1. It is very easy to use and maintain.
  2. It comes with JDK hence separate installation is not required.
  3. It is database independent driver hence migration from one database to another is easy.

- **Limitations:**
  1. It requires multiple conversion hence it is slower in performance.
  2. It depends on oDBC driver which work on Microsoft Windows operating system only.
  3. It is obsolete driver. No support from JDK 1.8 onwards.

JDBC Type 2 Driver Architecture



JNI : Calls C/C++ function into Java.

Type-2 Driver : Converts JDBC call into C/C++ Call which can understand by DB.

Typically it is provided by database vendor.

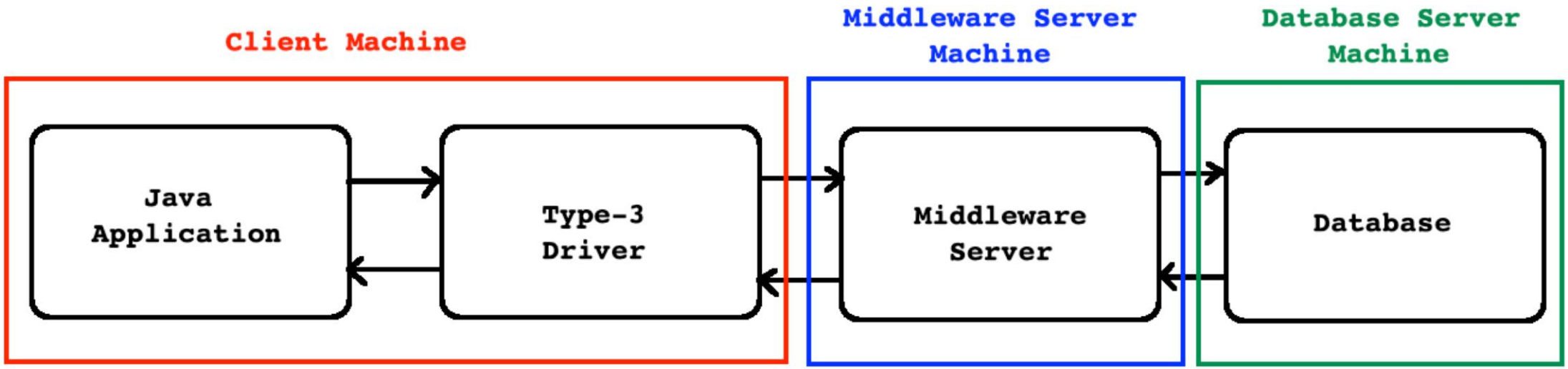
- Type-II driver is also called as Native API Driver.

- Example : Oracle Call Interface (OCI) Driver.
- *Note : MySQL do not support Type-2 Driver.

JDBC Type-2 Driver

- **Advantages:**
  1. It is based on vendor provided database specific native library hence ODBC support is not required.
  2. In comparison with Type-1 driver, it requires less call to communicate with database hence it is faster in performance.
  3. Available for Windows, Sun Solaris, Linux. Hence more portable than Type-1 Driver.
- **Limitations:**
  1. Since it is database dependant driver, migration from one database to another is difficult.
  2. It is platform dependant driver.
  3. It is necessary to install native library on client machine.
  4. All the database vendors do not provide native library.

JDBC Type-3 Driver Architecture



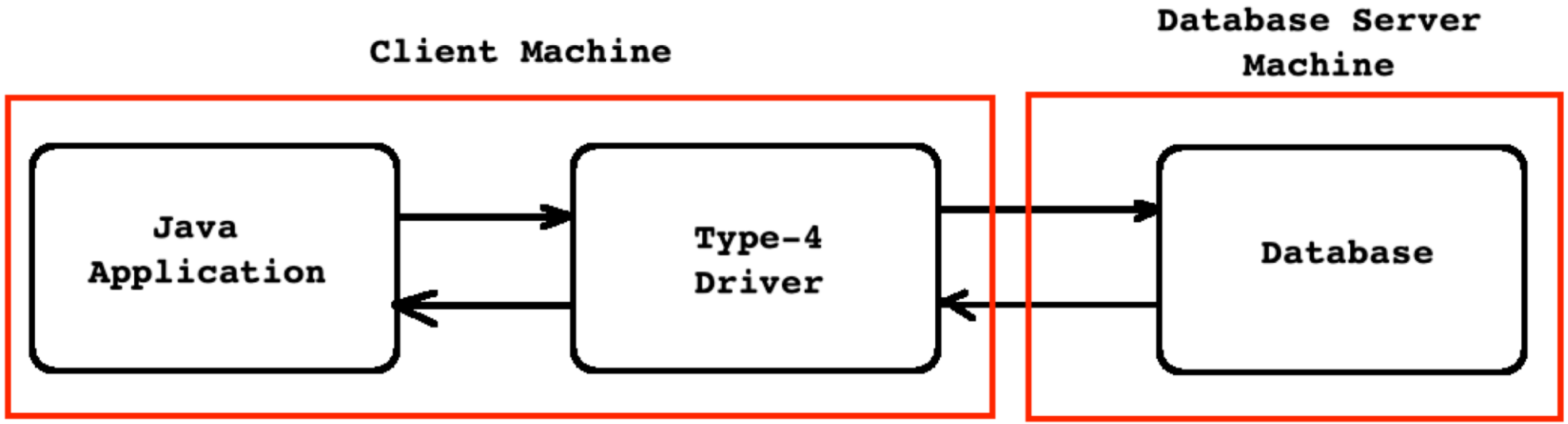
- Type-3 driver is also called as Middleware driver.
- It is the only driver, which is database independent and platform independent driver.
- Middleware server implicitly use Type-1/Type-2/Type-4 Driver to communicate with database.
- Type-3 driver follows 3 tier architecture.
- Explore : idssoftware.com
- JDBC client use socket to communicate with middleware server.

JDBC Type-3 Driver

- **Advantages:**
  1. It is a pure Java driver hence truly portable.
  2. No need to install oDBC driver or vendor provided database specific library on client's machine.
  3. Middleware server is a application server which helps us for auditing, load balancing or logging.

- **Limitations:**
  1. Network support is required on client's machine.
  2. It is costly to maintain.
  3. Database-specific coding is required at middleware server.

JDBC Type-4 Driver Architecture



- JDBC Type-4 driver is also called as Database Protocol Driver/Thin Driver /Pure Java Driver
- It is completely written in Java.
- This provides better performance than the type 1 and type 2 drivers as it does not have the overhead of conversion of calls into ODBC or database API calls.
- It directly communicate with database using vendor specific native protocol.

JDBC Type-4 Driver

- **Advantages:**
  1. Completely written in Java to achieve platform independence.
  2. It doesn't translate the requests into an intermediary format such as oDBC.
  3. No need of middleware server. Client application directly connects to database.
  4. Easily available.
  5. Simple to install.
  6. We can use it for standalone application as well as web application.
- **Limitations:**
  1. It is database dependant driver.

Steps to connect Java Application To Database

1. Include database (MySQL) connector into runtime classpath/build path.
2. Import java.sql package.
3. Load and register Driver.
4. Establish connection using users credential(username and password).
5. Create Statement/PreparedStatement/CallableStatement to execute query.
6. Execute query and process result.
7. Close resources.

Driver

1. Driver is interface declared in java.sql package.

2. Every driver class must implement this interface.

3. Driver implementation handles the communication with database.

4. When a Driver class is loaded, it should create an instance of itself and register it with the DriverManager. This means that a user can load and register a driver by calling:

   Class.forName(”com.mysql.cj.JDBC.Driver")

5. For more details please explore source code available in following file: mysql-connector-java-8.0.23/src/main/user-impl/java/com/mysql/cj/jdbc

MySQL Driver Implementation.

```
public class Driver
    extends NonRegisteringDriver
    implements java.sql.Driver {
    static {
        try {
            java.sql.DriverManager.registerDriver(new Driver());
        } catch (SQLException E) {
            throw new RuntimeException("Can't register driver!");
        }
    }
    public Driver() throws SQLException {
        // Required for Class.forName().newInstance()
    }
}
```

DriverManager

1. The DriverManager class is the traditional management layer of JDBC, working between the user and the drivers.
2. It keeps track of the drivers that are available and handles establishing a connection between a database and the appropriate driver.
3. In addition, the DriverManager class attends to things like driver login time limits and the printing of log and tracing messages.
4. Note that the javax.sql package, otherwise known as the JDBC 2.0 Standard Extension API, provides the DataSource interface as an alternate and preferred means of connecting to a data source. However, the DriverManager facility can still be used with drivers that support DataSource implementations.
5. For simple applications, the only method in the DriverManager class that a general programmer needs to use directly is DriverManager.getConnection.

```
Connection con = DriverMananager.getConnection( url, user, password );
```

Connection

1. It is an interface declared in java.sql package.
2. Connection implementation instance represents connection with database.
3. A connection session includes the SQL statements that are executed and the results that are returned over that connection.
4. A single application can have one or more connections with a single database, or it can have connections with many different databases.
5. Methods of Connection interface:
    1. Statement createStatement() throws SQLException
    2. PreparedStatement prepareStatement(String sql) throws SQLException
    3. CallableStatement prepareCall(String sql) throws SQLException

Statement

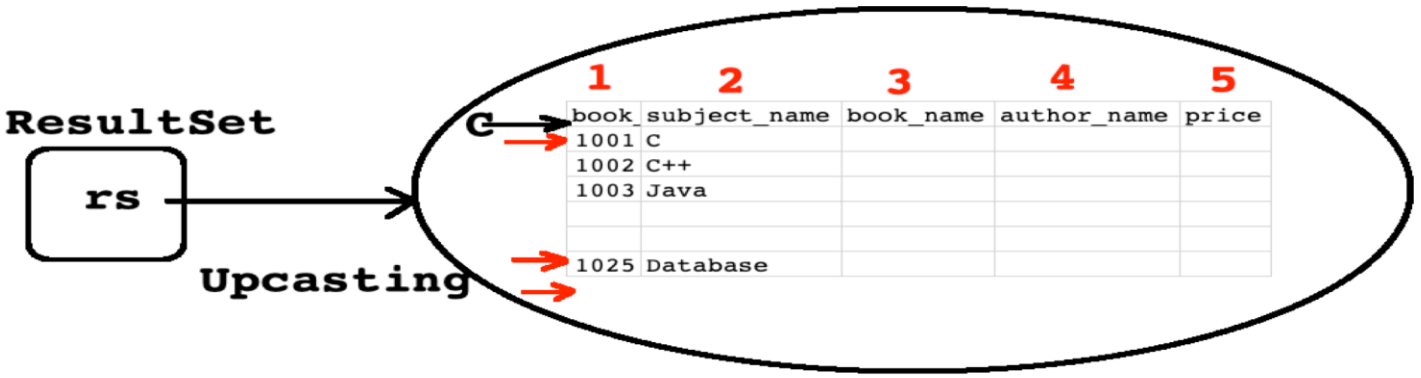
1. It is an interface declared in java.sql package.
2. A Statement object is used to send static SQL statements to a database.
3. The Statement interface provides basic methods for executing statements and retrieving results.
4. Methods of Statement interface:
    1. ResultSet executeQuery(String sql) throws SQLException
    2. int executeUpdate(String sql) throws SQLException
5. Creating Statement Object:

```
Statement stmt = con.createStatement( );
```

ResultSet

1. It is an interface declared in java.sql package.
2. A ResultSet is a Java object that contains the results of executing an SQL query. In other words, it contains the rows that satisfy the conditions of the query.
3. The data stored in a ResultSet object is retrieved through a set of get methods that allows access to the various columns of the current row. The
4. A ResultSet object maintains a cursor, which points to its current row of data. The cursor moves down one row each time the method next is called. When a ResultSet object is first created, the cursor is positioned before the first row, so the first call to the next method puts the cursor on the first row, making it the current row. ResultSet rows can be retrieved in sequence from top to bottom as the cursor moves down one row with each successive call to the method next.

ResultSet



Instance of ResultSet Implementation Class

```
stmt.executeQuery( String sql )
boolean status = rs.next( );
```

Thank You.

akshita.chanchlani@sunbeaminfo.com